CALIBRATION OF OPTICAL SEE-THROUGH HEAD MOUNTED DISPLAY WITH MOBILE C-ARM FOR VISUALIZATION OF CONE BEAM CT DATA

by

Tiffany Chung

A thesis submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Engineering.

Baltimore, Maryland

May 2017

© Tiffany Chung

All Rights Reserved

Abstract

This work proposes the visualization of Cone-Beam Computed Tomography (CBCT) volumes in situ via the Microsoft Hololens, an optical see-through Head-Mounted Display (HMD). The data is visualized in the display as virtual objects, or holograms. Such holograms allow the CBCT volume to be overlaid on the patient at the site where the data was acquired. This is useful in orthopedic surgeries due to the nature of the CBCT scans of interest. A known visual marker is tracked using both the Hololens and the RGBD (Red Green Blue Depth) optical camera rigidly mounted and calibrated to the C-arm. By combining the transformations, the calibration defines the spatial relationship between the HMD and the C-arm.

The calibration process and visualization have been confirmed and enable advanced visualization of CBCT data for orthopedic surgery. The hardware capabilities of the Hololens currently limit the quality of the volume rendered and the precision of the calibration. However, future improvements to each component have been identified and with the ever-improving technology of HMDs, this limitation is only a temporary barrier to clinical usage.

Advisor: Dr. Nassir Navab

Acknowledgments

Many thanks to Sing Chun Lee, Bernhard Fuerst, Javad Fotouhi, Alexander Barthel, Long Qian, and Dr. Nassir Navab of the Computer Aided Medical Procedures (CAMP) lab, and Dr. Russell H. Taylor of the Laboratory for Computational Sensing and Robotics (LCSR) at The Johns Hopkins University in Baltimore, Maryland for their endless patience and guidance.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Orthopedic surgeons rely heavily on the use of radiographic imagery to navigate the patient's anatomy and confirm hardware placement with minimal invasion. Traditional methods for certain procedures can require over 100 X-ray images to place one piece of hardware, exposing the patient and all medical staff in the operating room to an extensive amount of radiation over time. Current solutions to mitigate this render 2-dimensional X-ray images and 3-dimensional Cone-Beam Computed Tomography (CBCT) volumes on monitors inconveniently located around the operating room. This setup can make it difficult for surgeons to mentally transform structures in the images on the monitors to their corresponding physical locations in the patient. The spatial reasoning required to correctly and efficiently understand the information displayed on the monitors with the corresponding anatomy creates additional mental stress on the surgeon. This is mitigated if the surgeon directly views the information at its anatomical location on the patient. Additionally, a new X-ray must be taken each time the surgeon would like an updated view of the patient, or from a different angle, resulting in a large number of X-ray images and radiation per procedure. Acquiring these X-rays is also time-consuming and thus contributes to the duration of

the procedure, which correlates with increased risk of infection, time under anesthesia, and blood loss, among other concerns for patient safety.

1.2 Background

1.2.1 Transformations

Geometric transformations are used in augmented reality to describe the change in rotation and translation between different frames of reference. Rigid 3D transformations are most commonly represented by a 4×4 matrix made up of a 3×3 rotation matrix R and a 1×3 translation vector t. The combination of rotation and translation components in the transformation requires homogeneous coordinates and thus the fourth row is added to the matrix. The equation for this matrix is as follows:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

The rotation may also be represented by a Quaternion, by Unity's internal calculations in particular, to avoid Gimbal lock[1]. Rigid transformations are used in this project to calibrate and relate the position and orientation of the volume in the world space to the Hololens space.

1.2.2 CBCT Data

The CBCT data is formatted as a directory of DICOM files, or more intuitively a reconstructed volume of CBCT slices of a configured step size between cross-sectional slices. Each CT slice can be represented as a conventional grayscale image. The pixel values represent the relative radiodensity of the subject scanned, often normalized to

enhance specificity at the values of the tissues of interest. Unlike in traditional CT scans where grayscale values are defined in absolute Hounsfield units, the grayscale values of CBCT scans are defined relative to other values in the volume[2]. DICOM files can be easily converted to other image file types such as `.tif` or `.jpg`, which are easier to view and work with. Many applications such as Matlab[3] can perform this conversion. Each pixel has an (x, y) coordinate in the slice, and the slice number corresponds to the depth in the volume. Thus the pixels across all slices can be combined to form a volume of 3D pixels, or voxels. Different methods of visualization can view the volume from a point of view other than the one from which the original scan was acquired. 3D reconstructions can render the surface, the volume, or simply segment the image by thresholding the radiodensity values.

1.2.3 Visual Marker Tracking

Marker tracking is a computer vision technique used in augmented reality to calculate transformations between coordinate systems. A known marker in the real world is viewed by a camera in a known position relative to the real world. Thus, the transformation from the marker location to the camera is computed, and movement of the marker requires recalculation of the transformation. Vuforia[4] is an augmented reality Software Development Kit (SDK) with marker tracking capabilities and integration with the Unity game engine, which is used by this project.

1.2.4 Head-Mounted Displays

A head-mounted display (HMD) is an augmented reality display device, worn on the head of the user, to directly display projected images in the user's line of sight. The two types of HMD are "optical see-through" and "video see-through." Optical see-through HMD allows the user to view the real world through a transparent material onto which the augmented image is projected; while video see-through obtains live video of the real world, overlays the augmented information, and then displays the

combined image to the user in an internal display. The advantage of optical over video see-through is that in case of device failure, the surgeon can still view the real world without the augmentation. This is particularly advantageous for medical applications. A video see-through device will display only a black screen, effectively blinding the surgeon. Thus an optical see-through HMD was chosen with surgical applications in mind. However, optical see-through is known to introduce a lag in the display of augmented objects, which raises additional challenges in clinical applications. The Microsoft Hololens[5], a powerful optical see-through HMD, mitigates the concern for the lag. The augmented image is projected onto a glass display through which the user can still view the real world (see Figure 1.1). The Hololens outperforms other optical see-through HMDs in "contrast perception, task load, and frame rate"[6]. This HMD is, at the time of this project, state-of-the-art technology for developing augmented reality applications and was chosen for these properties.

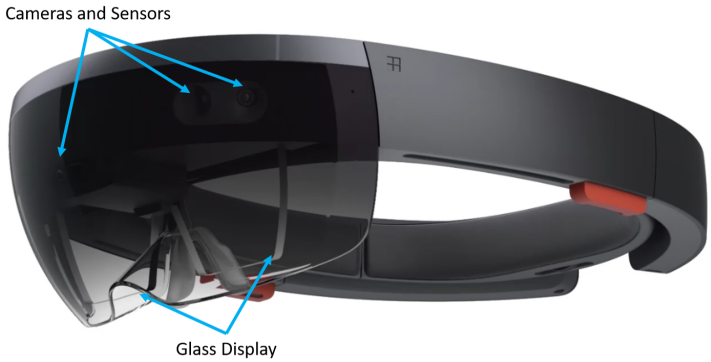


Figure 1.1: Microsoft Hololens.

1.2.5 Unity

Unity[7] is a multiplatform game engine with powerful graphics rendering, and Hololens and Vuforia integration. Each Unity project is made up of scenes such as a loading screen and each stage of a game. Objects in the scenes are types of GameObjects, such

as the camera, lights, cubes, and any other object present in the scene. GameObjects have properties such as Transform, Mesh, Textures, Shaders, Materials, and custom scripts. Custom scripts allow game logic to be added to the GameObjects. Unity is the development tool for Hololens applications in this project.

1.3 Related Work

A novel approach to calibrate an RGBD camera rigidly mounted to the C-arm produces the transformation from the CBCT volume to the RGBD camera[8]. This transformation allows visualization of live video from the RGBD camera to be overlaid on the CBCT data, providing surgeons with live feedback regarding their location relative to the CBCT volume. The transformation acquired by this system is one of the transformations necessary for the calibration in this project. This mixed-reality approach visualizes augmented views of the volume on a large external monitor.

Other CBCT visualizations in the Hololens simply render a CBCT slice viewer as a "screen" in the Hololens space with voice commands and hand gesture input[9]. These are similar to viewing a CBCT slice viewer on a conventional monitor controlled by computer mouse input. There are existing implementations of volume rendering CBCT data in Unity, but they are not designed to be deployed and run on the Hololens. These related viewers are proprietary, otherwise poorly and informally documented, and few truly render volumes.

1.4 Proposed Solution

The increased prevalence of HMD allows such visualization beyond flat monitors in 2D to augmented reality, such as 3D holographic images via the Microsoft Hololens. Thus, the CBCT volume can be visualized as a hologram in situ, at the same location

that the CBCT volume was taken by the C-arm, bypassing the need for the surgeon to mentally align the two spaces. Furthermore, visualization from any angle is possible without acquiring a new CBCT scan. The process to visualize a CBCT volume in the Hololens space is two-fold – a calibration of the Hololens to the C-arm, and the volume rendering of the CBCT data. By such a calibration, the visualization will remain in place as the surgeon moves around the space freely.

Chapter 2

Calibration Steps

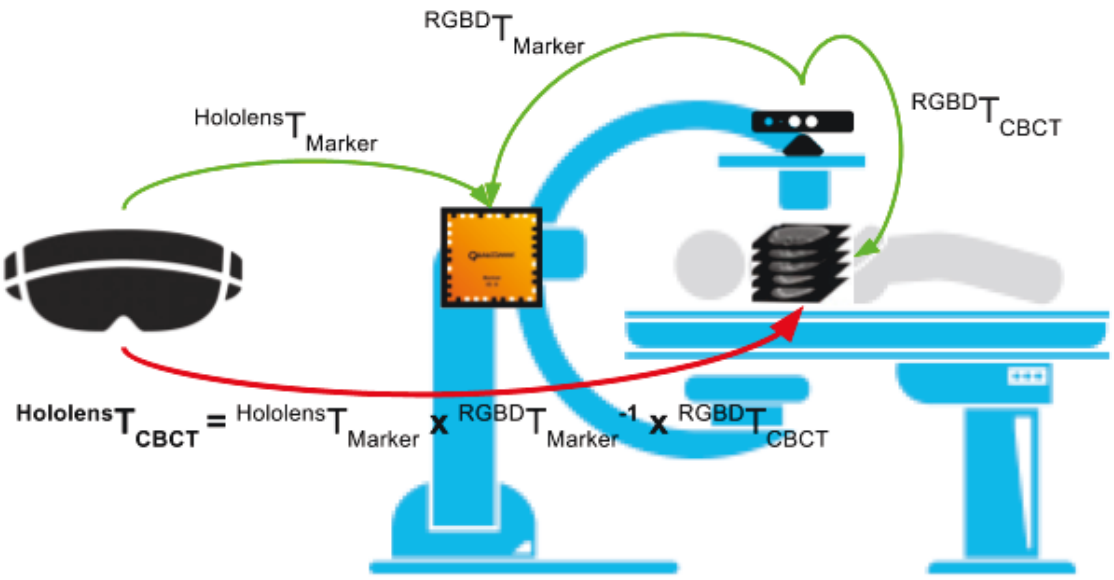


Figure 2.1: Calibration overview.

The three transformations from the Hololens to the Marker, RGBD camera to the Marker, then RGBD to the CBCT are each explained in more detail in the following subsections. The combined transformation from the Hololens to the CBCT is the desired calibration in Figure 2.1.

2.1 CBCT to RGBD Calibration

The calibration of the C-arm to the RGBD camera provides the known transformation from the CBCT volume to the RGBD camera given the fixed orientation of the optical camera to the C-arm[8]. The CBCT and Depth camera simultaneously scan the same object, and the surface points of both point clouds are registered using Iterative Closest Point[10] to determine the transformation between the CBCT and the RGBD camera. This information is directly loaded into the resulting transformation. Note that the CBCT data is in millimeter units.

2.2 RGBD to Marker Calibration

Vuforia's Unity plugin is used to compute and write the transformation from the RGBD camera to the marker to a file which is later accessed by the Hololens application. Information about the marker to be identified and its scale is inputted to Vuforia's configuration. This is a one-time computation because this implementation assumes that the C-arm will not move relative to the CBCT volume to be rendered.

2.3 Marker to Hololens Calibration

Vuforia's Unity plugin for Hololens development is used to track the same marker seen by the RGBD camera. Vuforia for Hololens has an "Extended Tracking" feature that allows the tracking to persist despite removal of the marker in conjunction with the Hololens mapping of the surroundings on initialization. Unity directly interfaces with the Hololens camera to obtain its transform and other necessary properties.

2.4 Hololens to CBCT Calibration

The combined transformation is the resulting transformation in Figure 2.1. Each of the transformations are applied in the specified order to achieve the transformation of where and how the volume will be rendered. The specified order is important, as transformations are applied by matrix multiplication, which is not commutative. The resulting transformation allows the CBCT data to be directly augmented on the patient's body.

Chapter 3

Volume Rendering

As the CBCT is a volume of data, viewing the volume is not as trivial as viewing a single image. For example, given a CBCT volume of the skull, the user can iterate through each 2D slice of the skull. However, if the user would like to view the skull as a 3D volume, from an angle different from the angle at which the scan was acquired, the information represented in each slice must be combined to reconstruct the whole skull, not just the surface, then recomputed and rendered from the desired view. This process is called volume rendering, and is subsequently explained in greater depth. Matlab is used to process the data, and Unity is used to visualize it.

3.1 Digitally Reconstructed Radiograph

A Cone Beam CT scanner acquires information using a cone-shaped beam, then digitally reconstructs the information into a volume of grayscale values, or a Digitally Reconstructed Radiograph (DRR). This processed data structure is more suitable for visualizing medical data and the volume of voxels can be directly used in volume rendering by ray casting.

3.2 CBCT Pre-Processing

The CBCT volume must be pre-processed in order to remove noise and improve visualization. The most efficient and effective pre-processing method for the bone model and more generally, all volumes, proved to be importing the DICOM slices into a DICOM viewer such as Imfusion[11] to maintain correct slice order, then exporting to a `.raw` file with metadata. Scattering noise artifacts commonly found in CBCT volumes can be removed by labelling the desired objects in the DICOM viewer then exported to a `.raw` file with metadata. This also allows consistency in axes and dimensions between the two volumes. Then, in `BoneFromRaw.m`, each file is read into a 3D array and manually rescaled to visible values. The labelled data used to select the objects of interest in the CBCT scan may be unnecessary with more intense pre-processing methods in the DICOM viewer.

3.3 Volume Versus Image Rendering

The DRR volume represents 3D data of voxels, but currently display monitors can only render 2D images of pixels. Methods such as ray marching (related to ray casting) are required to display a 2D projection of the 3D volume based on the point of view. Ray marching iteratively projects rays from the point of view through the volume to construct the 2D projection, as can be seen in Figure 3.1. The Hololens simulates rendering 3D "holograms" by continuously updating the 2D projection based on the point of view of the user as calculated by the camera on the Hololens. As the user moves in the world space, their perspective changes and the Hololens must continuously update the transformation of the hologram in the viewing plane relative to the user's current position and orientation.

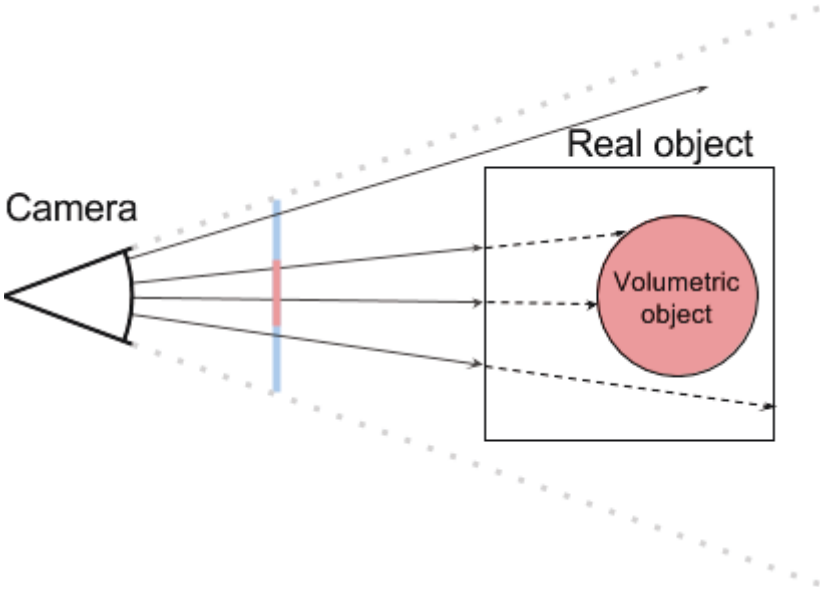


Figure 3.1: Rays casted from camera through scene intersecting with volume[12].

3.4 Ray Marching Algorithm

Ray marching is an iterative ray casting algorithm suitable for a 3D volume as the volume dimensions are known[12]. A 3D voxel volume is generated from the DICOM slices, then on each frame shaders iteratively cast rays from the camera location to each pixel in the current clipping plane and sum the values as the rays pass through the mesh to compute the 2D projection.

3.5 Unity Implementation

The ray marching implementation in Unity[1] is comprised of a Cube GameObject inside of which the volume is rendered, a Clipping Plane GameObject onto which the projection is rendered, and many shaders and scripts to compute the projection.

[1]This implementation is based on a publically available, unlicensed Unity project[13]. Many modifications to this code have been made, as the original code has little to no documentation and is not regularly or actively being developed. This implementation was chosen due to its ease of access and lack of alternatives.

3.5.1 Cube GameObject

A non-intuitive but crucial note is that the Cube GameObject itself should not be rendered by the Unity scene. `RayMarching.cs` and `SliceMesh.cs` attached to the camera GameObject will refer to the Cube GameObject in their "Cube Target" and "Meshes/Element" fields, respectively. Thus, only the volume and not the outer cube structure will be rendered.

3.5.2 Slice Importation

CBCT volumes typically have 256 slices; for ease of importation, `RayMarching.cs` loads an entire directory of slices from `Assets/Resources` into the project. Local files that need to be deployed to the Hololens should exist in `Assets/Resources` or else they will only be locally accessible by the Unity project. Unity automatically imports Textures when files are added to the project.

Additional steps to maintain proper formatting of the data through the steps and when written to file as a `.tif` image in Matlab required extensive research and troubleshooting with little help from Unity and Matlab documentation. Unity does not easily import any image format as a texture, and the resulting texture and volume visualized varies widely with different image formats and import parameters. These differences can be observed in the image metadata created when Unity imports images. When the `.tif` files are imported, they may or may not be read/write enabled. Another issue encountered from differing image formats across operating systems is that each of the 256 images may need to be trivially opened and closed in an image editor and then given read/write access in Unity on texture import in order for Unity to successfully load the slices. The `.tif` files do not come with alpha values, so on import the "Alpha Source" must be set to "From Gray Scale," and this can be confirmed by the updated Default Format of "RGBA DXT5" at the bottom of the

panel. It is imperative that the textures have alpha values for the proper projection to be computed. Without this step the Default Format will be "RGB DXT1," and the projection will incorrectly appear on the nearest face of the Cube GameObject as a 2D image. Also, the Ray Marching script that loads the slices orders the slices by file name, and this requires careful naming conventions when writing the slices to image files. The Matlab scripts for the bunny model preserve the original sequential file names, and the scripts for the bone model rename the files to shorter names while enumerating the slice number with leading zeroes if necessary to maintain order.

3.5.3 Volume Generation

The first step in volume rendering is generating the volume itself. This is done in `GenerateVolumeTexture()` of `RayMarching.cs`. As the CT volume acquired by this C-arm is 256×256 pixels per slice by 256 slices, this is a $256 \times 256 \times 256$ volume. Each pixel has value $[0, 255]$ and for volume rendering Unity requires RGBA color channels. For each slice, the pixel array is read into a `Texture3D` volume with the known dimensions. There is some additional logic for offsets to ensure the volume has valid dimensions.

3.5.4 Clipping Plane Calculation

A cube is a common mesh to contain a volume because the clipping plane formula for a cube is easy to compute and is feasible for any perspective of the camera. `SliceMesh.cs` and related scripts generate the triangles used to render the cube, and on each frame the cutting plane through the triangle mesh is calculated. It is on this plane that the Texture is placed as a Material. Note that Unity's units are in meters, while the C-arm used for this project outputs slices with units in millimeters. When computing the clipping plane, the scale of the volume must be scaled down by 1000 to reflect this.

3.5.5 Shaders

The shaders are located in `Assets/Shaders` and in particular the ray marching is done in `Ray Marching.shader`. `raymarch()` is called for each ray emanating from the camera to each texture coordinate. Each ray is "marched" through the volume, accumulating value at each step count until the ray passes into the surface of the volume[12]. As the volume in this project is not a simple geometric object, each ray must be marched through the full depth of the volume. This simple accumulation function can be replaced and improved by more sophisticated transfer functions to select specific anatomy to be visualized, and enhance overall visualization.

3.5.6 HMD Display

A GameObject with a child Cube GameObject with a child Clipping Plane GameObject manipulated by the computed transformation displays the 2D projection of the volume. The calibrated transformation from the Hololens camera to the clipping plane is constantly updated by the Hololens to reflect the user's point of view in `ManipulateCube.cs`. The transformation is only set on `Start()` as the volume should not move relative to the fixed marker in the world space as the user moves about the world space. Once the marker is tracked and the volume is rendered, the marker can be removed and the Hololens will use "Extended Tracking" to persist the visualization in that position relative to the world space environment. However, if the marker is moved, then the calibration will be incorrect and the Hololens will rerender the volume at the incorrect transformation.

3.5.7 Logging and Heads-Up Display

Heads-Up Display (HUD) text is displayed in the upper-right-hand corner of the HMD display using a Unity UI GameObject to update the user on the system process. In Figure 3.2, the "Ready" message indicates that the volume is ready to be rendered.

The same messages are logged in the Unity Debug Log. The application may require several minutes to initialize, so the help text lets the user know when the application is ready to track the marker, or if the application failed due to some error.

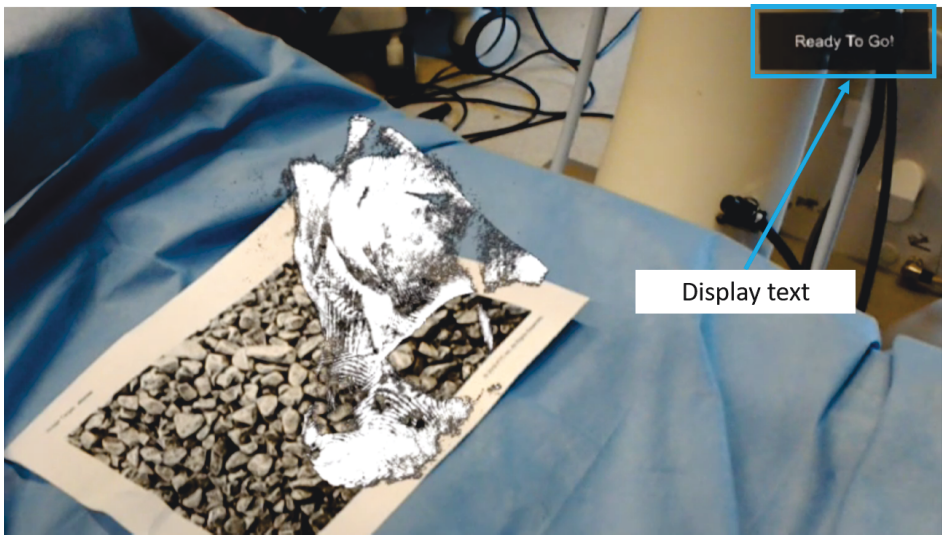


Figure 3.2: "Ready" message in heads-up display.

Chapter 4

Method

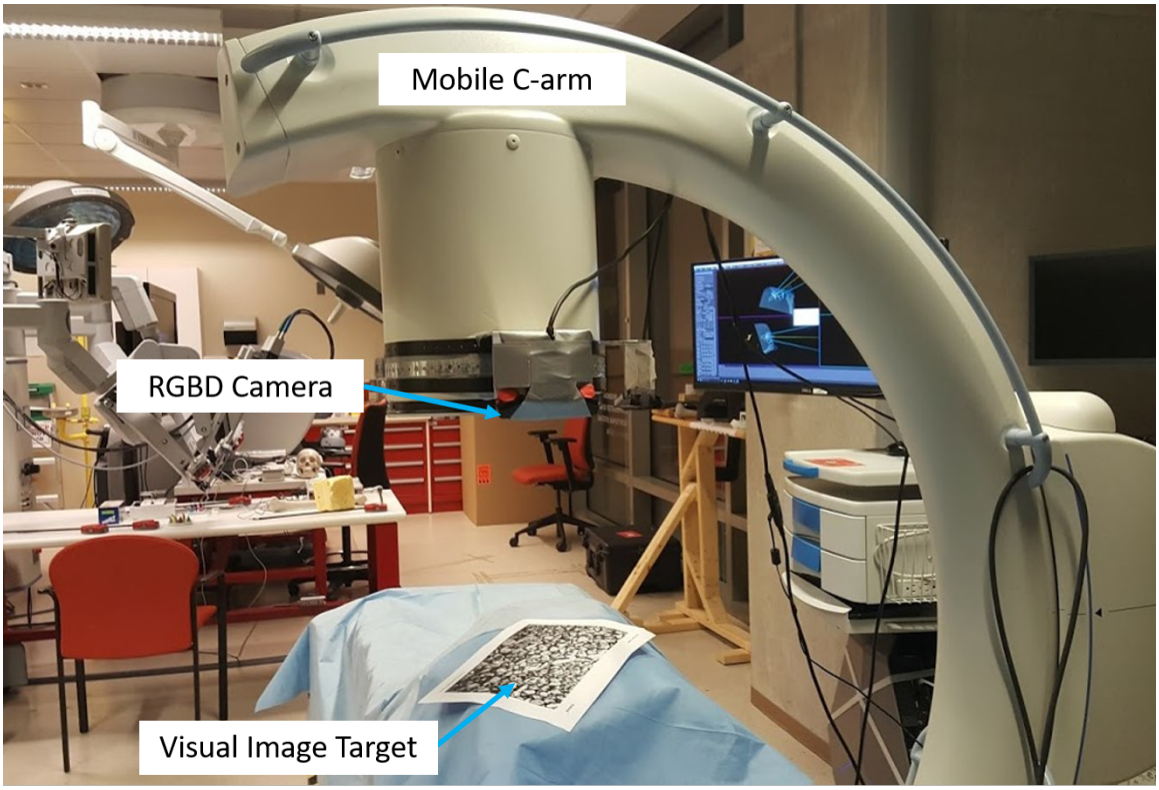


Figure 4.1: C-arm, RGBD camera, and visual marker setup.

The full implementation of this project begins with pre-processing CBCT data. The system is set up as shown in Figure 4.1, then two Unity applications are run sequentially to visualize the CBCT data in the HMD space.

4.1 CBCT Data Pre-Processing

Run the Matlab script with desired pre-processing algorithm `DICOMMODEL_ToTiff_ALG.m` or `BoneFromRaw.m` to threshold the CBCT data to desired intensities. This will reduce noise and remove scattering artifacts of the CBCT. The directory of the pre-processed CBCT data should be placed in `Assets/Resources` in order to be accessible by the Hololens.

4.2 System Setup

4.2.1 Tracking Camera

The marker to track is configured in the Vuforia configurations. Vuforia tracks the marker using a given AR camera GameObject in Unity, which can be a USB optical camera or Hololens webcam. In this implementation, an Intel RGBD optical camera is connected over USB to compute the RGBD to marker transformation, and the Hololens camera is used to compute the Hololens to marker transformation.

4.2.2 Image Target

This project uses Vuforia's proprietary marker, the "stones" image target seen in Figure 4.1. Vuforia requires image targets to be uniquely identifiable by orientation and the pixel quality must be high enough to support recognition from nontrivial distances.

4.3 Transformation Application

In Vuforia's scripts, `ImageTargetBehaviour.cs` has two functions that handle when the marker is found and when the marker is lost, respectively. In `RGB_To_Marker`, when the marker is found, the transformation is actually from the marker to the

camera, and this is written to file. Thus in `CBCT_To_Hololens`, the transformation is applied in the same way. In the code level of implementation there is no need to compute the inverse, as the way the transformation is acquired by Vuforia is already from the marker to the camera, as desired.

In `CBCT_To_Hololens`, the `ManipulateCube.cs` script is applied to a GameObject that contains the Cube GameObject of which the volume is rendered. This GameObject is a child of the Vuforia ImageTarget GameObject so that the transformation will be applied relative to the location of the marker. `ManipulateCube.cs` applies each of the transformations in order, starting with marker to RGB camera, then RGB to Depth camera, and finally Depth camera to CBCT.

4.4 Obtain RGBD Transformation

Place the "stones" marker in a location visible to the RGBD camera and visible to the Hololens as seen in Figure 4.1. Do not move this marker. Connect the RGBD camera to the computer and open the `RGB_To_Marker` Unity application. In "Vuforia Configuration," select the optical camera as the "Webcam". Run the application and confirm that the RGBD camera obtained the position of the marker by checking that the "distance to the marker" is greater than 0.0 and the transformation has been written to `CBCT_To_Hololens/Assets/Resources/transformation_RGB.txt`. This file path can be changed in the Vuforia script, but the transformation file must be in this directory in order for `CBCT_To_Hololens` to have proper access.

4.5 Run Hololens Application

Open the `CBCT_To_Hololens` Unity application in Unity. In the HololensCamera GameObject, enter the name of the directory of the pre-processed CBCT data, and

confirm that the directory is in `Assets/Resources`. Turn on the Hololens. Build and deploy the Unity application to the Hololens device. It may take a few minutes for the application to begin in the Hololens; wait until "Ready!" displays in the HUD. Once the slices are loaded, find the marker with the Hololens. The volume will load momentarily. Once the marker is found and the hologram visualized, the marker can be removed. Take care that the Hololens does not view this marker again during this session, otherwise the transformation will be incorrectly recalculated to the new position of the marker. If the marker needs to be moved, the RGB to marker calibration must be run again to update the transformation.

Chapter 5

Experiments and Results

5.1 Preliminary Volume Rendering in HMD

Volume rendering in Unity is not novel; however, volume rendering in an HMD requires additional development due to the computational limitation of the Hololens. With significant research and implementation, this project confirms that CBCT volumes can be visualized by the Hololens frame rates ranging from 4 - 10 fps. The first volume rendered was an abdominal CT scan in 20 slices, seen in Figure 5.1.

Figure 5.1: Sample abdominal CT hologram, 2.973877 FPS.

5.2 Calibration Outcome

The calibration was verified independently of the volume visualization by manipulating the transformation of a virtual Teapot GameObject. Computing the transformation of the marker in the RGBD camera was straightforward and easily verified. Units in Unity and Vuforia were confirmed to be in meters, and the other transformations were scaled to meters also. The transformation was applied to the teapot in the Hololens and it was quickly confirmed that the teapot was at the location of the RGBD camera. Then the full transformation was applied and confirmed to be in the expected location of the CBCT origin. Note that the frames per second are at 30 in Figure 5.2, which is set by Vuforia during marker tracking. The maximum frames per second is 60 in the Hololens.

Figure 5.2: Teapot hologram to confirm transformation, 30.025552 FPS.

5.3 CBCT Volume Analysis

5.3.1 Sample Model

The first CBCT volume of 256 slices rendered was a CBCT scan of a 3D-printed Stanford bunny model. This model was chosen due to its simplicity of one material and few features. Noise in the scan was filtered out, but visualization was poor, if rendered at all. Upon manual inspection of the images imported, the grayscale values were too low and needed to be normalized. Binary thresholding was done on the images and this improved the visualization significantly, as shown in Figure 5.3. Note that the frames per second decreased dramatically, but the visualization is clear.

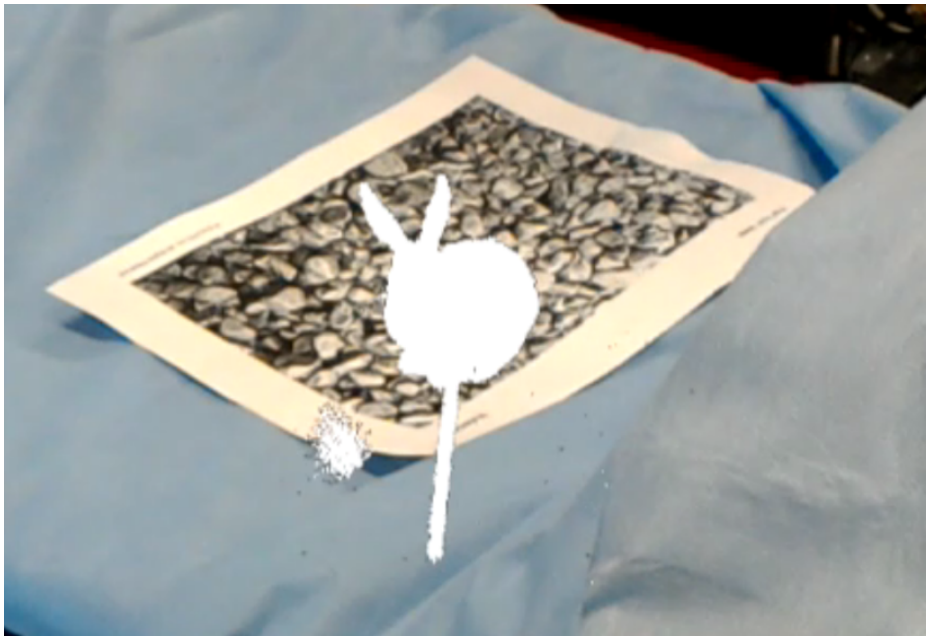


Figure 5.3: Bunny CBCT hologram, 3.349351 FPS.

5.3.2 Anatomical Model

Visualizing a more complicated anatomical CBCT scan of the bone model in Figure 5.4 was attempted, but the dominating presence of noise from scattering requires ad-

ditional pre-processing. Matlab scripts accompanying this project are used to convert a DICOM directory of CBCT slices or a `.raw` file from DICOM data to a directory of `.tif` images with no compression, 8 bit depth, and the ability to pre-process the volume. Many different methods were evaluated on different CBCT volumes to conclude proper compatibility with Unity and the best visualization. Their results are shown in Figure 5.5 and the processing details are discussed in the next section.

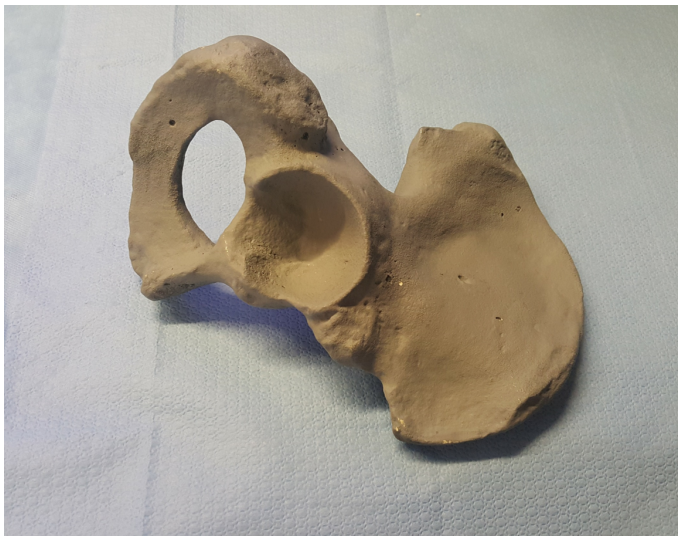


Figure 5.4: Anatomical hip bone CBCT model.

5.4 CBCT Pre-Processing

5.4.1 Histogram Analysis

Preliminary information about each volume is acquired by the Matlab scripts with suffix `_Hist`, which simply read in the DICOM data and display histograms of the values using the Matlab function `imhist(Volume)`, as seen in Figure 5.5. This allows for better understanding of the distribution of values of the desired volume versus the noise, and where to assign thresholds.

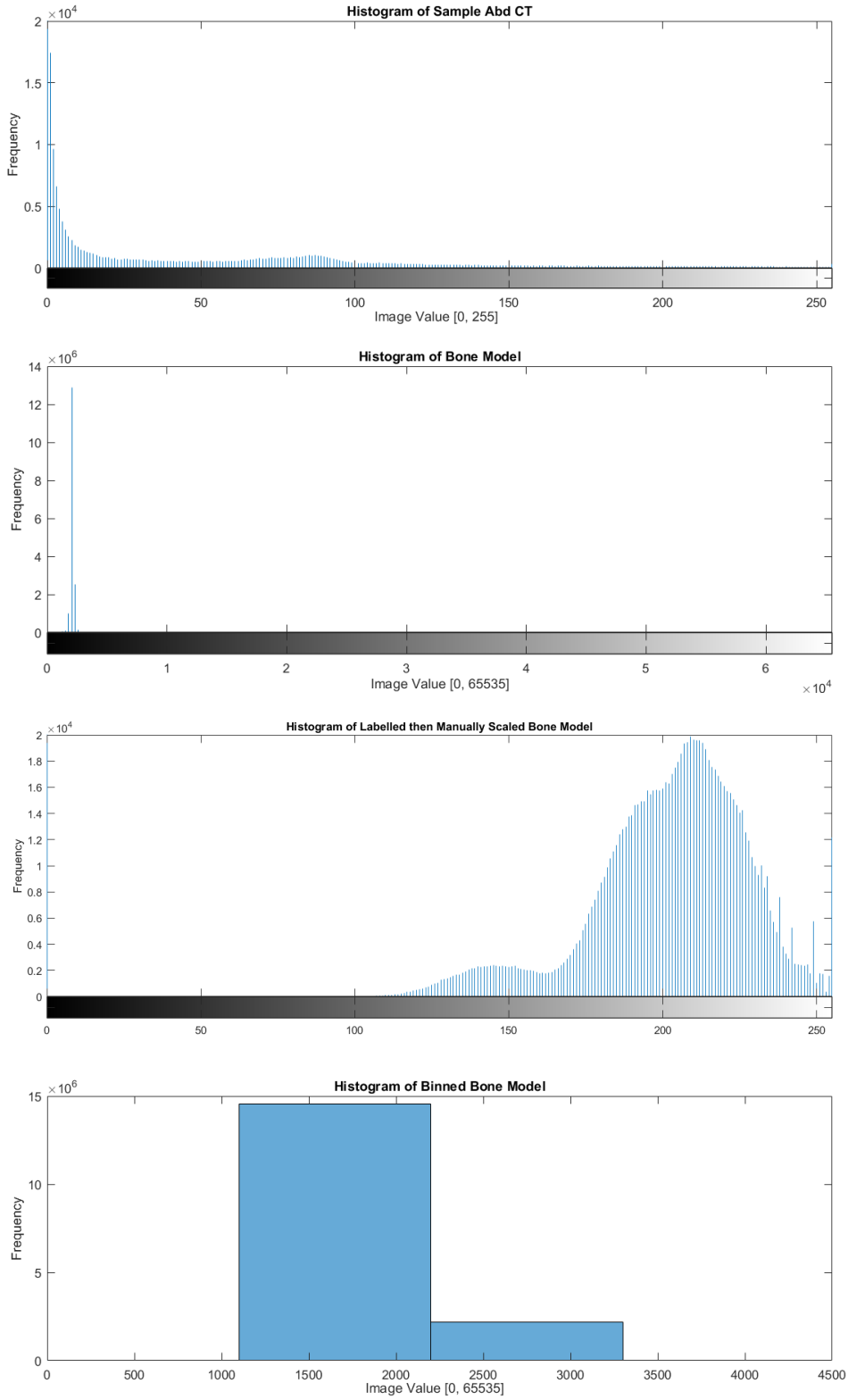


Figure 5.5: Histograms of CT volumes.

5.4.2 Binary Thresholding

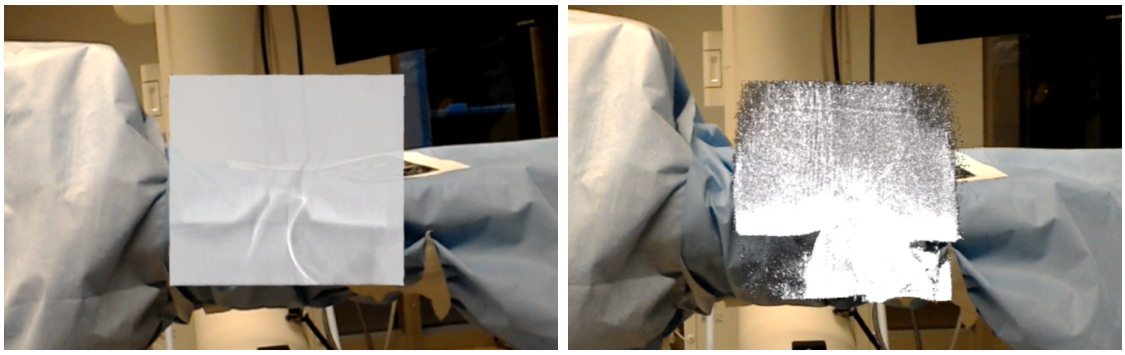
Binary thresholding is not suitable for this anatomical bone model because the image values correspond to relative radiodensity values, which may provide important information for the surgeons.

5.4.3 Normalization by Scaling Maximum Value

`DICOMBoneToTiff_ManualRescale.m` tracks the minimum and maximum values in the volume, then scales all the image values to be within this new range. The bone model DICOM images are of 16-bits for a range of $[0, 65536]$, but the histograms in Figure 5.5 showed that the image values were actually in the range $[2000, 4000]$. See the visualization in Figure 5.6. The visualization is very poor because all of the scattering from noise is in the same range as the bone model, and everything, including the noise, is scaled up.

5.4.4 Normalization by Histogram Bins

The Matlab function `histogram()` computes a configurable number of bins with equal count, as can be seen used in `DICOMBoneToTiff_HistBins.m`. The image values of each slice are thresholded down to the nearest bin value. This script is configured with the number of "bins" that corresponds to the number of different radiodensities, which in turn represent the different materials of the model. Finally, the image values are rescaled to the maximum value in the entire volume to adapt to and improve the range of visibility. Unfortunately, CBCT volumes acquired by the C-arm are known to contain noise artifacts of scattering, which dominates the histogram frequency. It was confirmed in the histograms (see Figure 5.5) that the values of the data of interest are higher than the values of the scattering, but with a far lower frequency, thus histogram binning is not a suitable method for the bone model. This method is only a slight improvement to normalization by scaling to the maximum value.



(a) Bone hologram scaled by max value. (b) Bone hologram normalized by bins.

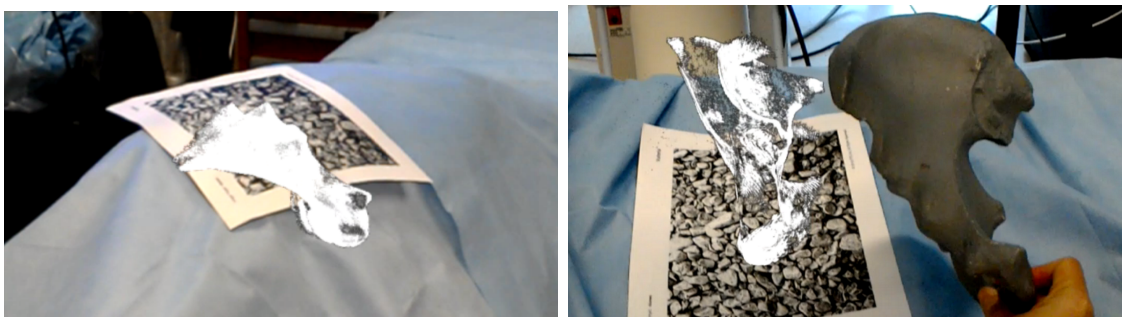
Figure 5.6: Bone CBCT normalization results.

5.4.5 Normalization by Labelling

To address the scattering noise directly, the bone in the volume is manually labelled in a DICOM viewer and exported as a `.raw` file with voxel values of 1 for bone, and 0 otherwise. This is read into Matlab in `DICOMBoneToTiff_ManualRescale.m` as a 3D array. Note that the Matlab script requires an additional step to orient the 3D array in the correct order of axes. Then, for each slice of the original volume, the values are multiplied by the corresponding labelled slice. This is a logical AND operation that preserves the original values if that voxel in the labelled volume has a value of 1. Finally, the slices are scaled to the range of the maximum value in the volume. The result can be visually confirmed by inspecting all the slices.

5.4.6 Normalization by Manual Scaling

The normalization by labelling method solved the problem of the scattering noise, but the scaling shifts the image values to be too high and thus the relative radiodensity information is lost. The CBCT volume was manually scaled by an exponential transfer function to preserve the relative grayscale information. Also, the scattering noise was manually removed by thresholding the known frequencies, effectively labelling the desired bone model. This results in the best visualization in terms of reduced noise, relative radiodensity, and complex model.



(a) Bone hologram after labelling. (b) Bone hologram after manual scaling.

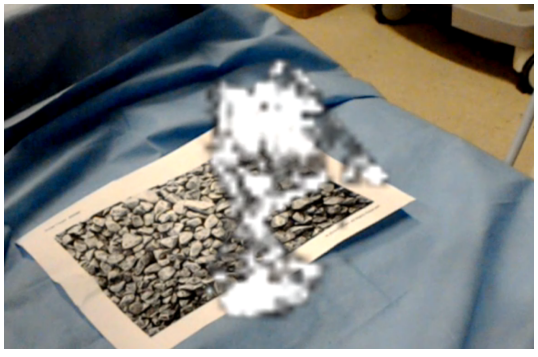
Figure 5.7: Bone CBCT normalization final results.

5.5 Frames Per Second Evaluation

There is a noticeable lag in the visualization when the user moves too quickly. While the volume is being rendered, Hololens Systems Processes reports an average framerate of 3 - 5 frames per second. The Hololens is capable of up to 60 fps, but Vuforia sets the frame rate to 30 fps. Downsampling of the volume increases the frame rate, but the tradeoff in visualization quality favors high resolution for low frame rate. See comparison in Figure 5.8.

(a) Bone hologram not down-sampled, 4.149326 FPS.

(b) Bone hologram down-sampled by 2, 6.787919 FPS.



(c) Bone hologram down-sampled by 8, 10.266868 FPS.

Figure 5.8: FPS of bone model with down-sampling.

Chapter 6

Discussion and Conclusion

6.1 Error Analysis

The "known transformation" of RGBD to C-arm has accuracy of 2.58mm[8], thus not the main source of error in the overall calibration. Error from the Vuforia marker tracking by the RGBD camera can be reduced by retrieving the average of transformations over the last 10 frames. The accuracy of Vuforia marker tracking by the Hololens is quite stable when confirmed visually, particularly when combined with "Extended Tracking" by the Hololens. This transformation was isolated and visually confirmed. The largest source of error is the instability of the computationally expensive volume rendering by the Hololens. The more quickly the user changes position, the greater the lag of the Hololens while recalculating the volume rendering.

6.2 Clinical Application

This project has received high-level positive support for clinical application – in particular, the visualization in an HMD instead of on multiple monitors in the operating room. However, processing limitations of the Hololens require more efficient and higher-quality visualization before legitimate clinical testing.

6.3 Future Work

6.3.1 Volume Rendering

The ray marching algorithm implemented in this project is not the optimal volume rendering algorithm. For example, Fast Fourier Transform volume rendering can dramatically improve visualization and rendering performance.

The volume is currently loaded, generated, rendered, and visualized by the Hololens in real time. However, the Hololens has streaming capabilities such that the computations for volume rendering can be done remotely. The Hololens sends the current transformation to a remote computer which loads and processes the volume, computes the current 2D projection based on the given transformation, then returns just the 2D projection for the Hololens to render. It is possible that a highly efficient volume rendering algorithm may be less computationally expensive than bidirectional streaming of data. The positive correlation between down-sampling and increased frame rate suggests that slightly decreased resolution still allows for a clear visualization, while improving frame rate to improve stability of the visualization.

6.3.2 Marker Tracking

Vuforia is a proprietary marker tracking SDK that can be replaced with a more lightweight and precise marker tracking program. Additionally, the Vuforia for Unity plugin only allows one camera to track the marker at a time, thus the project requires two separate Unity applications. A marker tracking program that concurrently tracks the marker in the RGBD optical camera and the Hololens would only require one Unity application and support continuous marker tracking. This also allows for C-arm movement and continuous recalibration.

6.3.3 Marker Location

The rigidly mounted RGBD optical camera on the C-arm system is the limiting factor for where the marker can be placed. However, if by a new method, the transformation from the CBCT volume in C-arm space to the marker in real world space can be calibrated, then the RGBD camera and its calibration is unnecessary. Then the marker can be placed anywhere on the C-arm as long as the transformation from the marker to the CBCT volume in C-arm space is known. The only marker tracking would be by the Hololens of the marker on the C-arm.

Appendix A

Program Documentation

A.1 Image Pre-Processing Matlab Scripts

Scripts/VolumeProcessing/README.md

```
# Matlab Scripts for Image Processing
Tiffany Chung (tchung12@jhu.edu)
Used with Visualization of CBCT Volumes in Hololens
April 2017

## Usage
This pre-processing of DICOM to `.tif` files is necessary to
import them as a resource in the Unity project for Ray Marching.
DICOM slices are of 2^16 bit depth giving a range of values
[0, 65536). The Matlab scripts iterate over `.raw` files or
over all `.IMA` files in the directory and converts them to
`.tif` files in a new directory with a suffix consistent with
the script. Each script converts with a different technique
to improve visualization for that model.
Note that Unity imports these images as textures with 2^8
bit depth which gives a range of values [0, 256).
Before using each script, verify the directory paths and that
the destination directory exists.
```

```
## BoneFromRawNew
Convert Bone `.raw` file to directory of `.tif` slices
in `BoneFromRawNew`. The imported `.raw` file already has
scattering noise removed and has been exponentially scaled.
The 3D array is scaled to the maximum value of the volume,
then written to slices.

## BoneFromRaw
Convert Bone `.raw` file to directory of `.tif` slices
in `BoneFromRaw`. Composite with pre-computed labelled `.raw`
file then manually rescale to range `[2000, 4000]`.
This is the preferred method because the axes and dimensions
are guaranteed to be consistent.

## **DICOMBoneToTiff_UseLabels**
Convert Bone DICOM volume to directory of `.tif` slices with
suffix `_LABELLED`. Using pre-computed labelling of
the bone model in `BoneDICOMLabelled/LabelMap.raw`, a logical
AND of the DICOM slice and the labelled slice is computed and
the original value is preserved, if the value is in the
labelled bone. Finally, the images are rescaled as in
`DICOMBoneToTiff_ManualRescale` in the range `[2000, 4000]`.
Note that the file names are of the form `Bone_###.tif`, with
leading zeroes if necessary because the RayMarching script
orders the slices by file name.

## DICOMBoneToTiff_BinThresh
Convert the labelled slices of the Bone volume to directory
of `.tif` slices with suffix `_BW`. Default binary
thresholding using `imbinarize` on the bone model.

## DICOMBoneToTiff_HistBins
Convert Bone DICOM volume to directory of `.tif` slices
```

```
with suffix '_BINS'. Normalize values to 4 bins according
to the histogram of the volume.
See images for histograms of different numbers of bins
for the Bone volume.

## DICOMBoneToTiff_ManualRescale
Convert Bone DICOM volume to directory of '.tif' slices
with suffix '_MANUAL'. Normalize values to a manually
configured threshold to remove the noise from the
scattering for the Bone volume.

## DICOMBoneToTiff_Rescale
Convert Bone DICOM volume to directory of '.tif' slices
with suffix '_RESCALE'. Normalize values to max value
over all images, which is about '4100'.

## DICOMBunnyToTiff_BinThresh
Convert Bunny DICOM volume to directory of '.tif' slices
with suffix '_TIFF_BW'. Binary thresholding to threshold
configured for Bunny.

## DICOMBone_Hist
Load the Bone DICOM volume and display the histogram of
values.

## DICOMBoneLabelled_Hist
Load the labelled slices of the Bone volume and display
the histogram of values.

## DICOMSampleCT_Hist
Load the SampleCT volume from the Unity RayMarching
example and display the histogram of values.
```

A.2 RGB to Marker Unity Project

Scripts/RGBToMarker/README.md

```
# RGB_To_Marker
Unity project with Vuforia plugin for RGB camera to track
the Vuforia marker.
Tiffany Chung (tchung12@jhu.edu)
April 2017

## Usage
1. Open project in Unity 5.4+ (this project used Unity 5.6).
Open Unity scene file ‘CubeScene‘.

2. Set the correct path in
‘Assets/Vuforia/Scripts/ImageTargetBehaviour.cs‘ to the
desired directory for the transformation to be written to.
If this is used with the ‘CBCT_To_Hololens‘ project, the
path should be to ‘CBCT_To_Hololens/Assets/Resources‘.

3. Select the RGB optical camera to use in
‘Assets/Resources/VuforiaConfiguration‘, scroll down to
‘Webcam‘ and set ‘Camera Device‘.
Leave ‘Disable Vuforia Play Mode‘ unchecked.

4. Place the Vuforia ‘stones‘ marker in the view of the RGB
optical camera. If another marker is used, this must be
configured in the ‘ImageTargetBehaviour‘ of the
‘ImageTarget‘ GameObject in the scene.

5. Run the scene in Play mode by clicking the Play button,
and confirm that a white cube appears at the center of the
marker and in the debug log the distance to the marker
appears. This indicates that the marker has been found.
Press the play button again to exit Play mode.
```

```
## Implementation
* `DefaultTrackableEventHandler.cs`
Contains `OnTrackingFound()` and `OnTrackingLost()` methods.
When the marker is found, a white cube appears at the
center of the marker.

* `ImageTargetBehaviour.cs`
Modified to write distance to target in debug log and if
distance is greater than 0, write the transformation (from
camera to marker) to `transformation_RGBD.txt` file to be used
by `CBCT_To_Hololens`.

* `transformation_RGBD.txt`
Text file containing the transformation.
First line is Unity Vector3 of position and second line
is Unity Quaternion of rotation.
```

A.3 CBCT to Hololens Unity Project

Scripts/CBCTToHololens/README.md

```
# CBCT_To_Hololens
Unity project with Vuforia plugin to visualize CBCT volume
in Hololens.
Tiffany Chung (tchung12@jhu.edu)
April 2017

## Usage
1. Open project in Unity 5.4+ (this project used Unity 5.6).
Open Unity scene file ‘Vuforia-2-Hololens‘.

2. Verify two transformation files exist.
    * ‘Assets/Resources/known_transformation.txt‘ is the
    transformation between the RGBD camera and the CBCT.

    * ‘Assets/Resources/transformation_RGB.txt‘ is the
    transformation between the RGBD camera and the marker.

3. Do not move the Vuforia ‘stones‘ marker from the
**RGBD_To_Marker** calibration. This should be in a
location visible to the Hololens.

4. Import the directory of the pre-processed CBCT volume
to visualize into ‘Assets/Resources‘. See Matlab scripts
to pre-process the DICOM data to ‘.tif‘ slices. Let Unity
properly import the textures. Then select all of the images
and set ‘Read/Write Enabled‘ to ‘true‘. The rest of the
settings should match ‘README_ImportSettings.png‘. Note
that this process is nontrivial and must be completed
exactly as instructed! Many hours of troubleshooting went
into determining this correct configuration.
```

5. Select the directory of volume in the ‘HololensCamera‘ GameObject under the "Path To Slices" of the ‘Ray Marching (Script)‘. Note that this directory must be in ‘Assets/Resources‘ in order for the Hololens to access once the application is deployed. The name of the directory does not need the ‘Assets/Resources‘ prefix, and should end with a trailing backslash.

6. Build the project with standard Hololens build configuration. ‘Unity C#‘ must be set to true.

7. Open the Visual Studio Project ‘App/CBCT_To_Hololens.sln‘. Set the Build, Release, Device, etc. parameters as necessary, and run with ‘Start Without Debugging‘.
It will take a few minutes to deploy.

8. Once the "Made With Unity" screen appears, wait another few minutes for the volume to be loaded and for Vuforia to be initialized.

9. View the marker in the Hololens, and the volume soon will be rendered.

10. To improve the stability of the visualization, once the marker is tracked, remove the marker and keep it out of sight of the Hololens. Thus the Hololens will not need to recompute the tracking each time the marker is viewed. The Hololens "Extended Tracking" feature will maintain the current position of the marker until the marker is found again.

Implementation

* ‘ManipulateCube.cs‘

Custom script attached to CubeVolume that reads the

transformation in ‘transformation.txt‘ and applies the transformation.

* ‘transformation_RGB.txt‘
Text file contains the transformation.
First line is Unity Vector3 of position and second line is Unity Quaternion of rotation.

* Ray marching scripts and shaders in ‘Assets/Scripts‘ and ‘Assets/Shaders‘ are from another project was imported.
The ray marching scene can be seen in the Unity scene file ‘sceneCube‘ but the pertinent parts have been copied to ‘Vuforia-2-Hololens‘.

References

[1] *Unity Documentation - Quaternion*. Unity Technologies. 2017. URL: https://docs.unity3d.com/ScriptReference/Quaternion.html.

[2] Tahmineh Razi, Mahdi Niknami, and Fakhri Alavi Ghazani. "Relationship between Hounsfield Unit in CT Scan and Gray Scale in CBCT". In: *Journal of Dental Research, Dental Clinics, Dental Prospects* 8.2 (June 2014), pp. 107–110.

[3] *MATLAB Documentation*. MathWorks, 2017. URL: https://www.mathworks.com/help/index.html.

[4] *Vuforia SDK*. PTC Inc. 2016. URL: https://developer.vuforia.com.

[5] *Microsoft Hololens*. Microsoft. 2017. URL: https://www.microsoft.com/microsoft-hololens/en-us/developers.

[6] Long Qian, Alex Barthel, Alex Johnson, Greg Osgood, Peter Kazanzides, Nassir Navab, and Bernhard Fuerst. "Comparison of optical see-through head-mounted displays for surgical interventions with object-anchored 2D-display". In: *International Journal of Computer Assisted Radiology and Surgery* (Mar. 2017).

[7] *Unity Manual*. Unity Technologies. 2017. URL: https://docs.unity3d.com.

[8] Sing Chun Lee, Bernhard Fuerst, Javad Fotouhi, Marius Fischer, Greg Osgood, and Nassir Navab. "Calibration of RGBD Camera and Cone-Beam CT for 3D Intra-operative Mixed Reality Visualization". In: *International Journal of Computer Assisted Radiology and Surgery* 11.6 (June 2016), pp. 967–975.

[9] Imran Shafiq. *Medical Imaging - DICOM volume rendering on hololens*. 2016. URL: https://www.youtube.com/watch?v=TogSmPaVoOc.

[10] P.J. Besl and Neil D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14 (2 Feb. 1992), pp. 239–256.

[11] *ImFusion Suite*. ImFusion, 2017. URL: http://www.imfusion.de.

[12] Alan Zucconi. *Volumetric Rendering Tutorial*. 2016. URL: http://www.alanzucconi.com/2016/07/01/volumetric-rendering.

[13] Brian Su. *Unity Ray Marching*. 2017. URL: https://github.com/brianasu/unity-ray-marching/tree/volumetric-textures-depth.

Curriculum Vitae

Tiffany Chung was born on June 16, 1995 and is from San Francisco, CA. She received her B.S. degree in Computer Science with a minor in Computer Integrated Surgery and M.S.E. degree in Computer Science from The Johns Hopkins University in Baltimore, Maryland in May 2017. Following graduation, she will be working as a software engineer in San Francisco, CA.